$1.20

Apple Assembly Line

Volume 2 -- Issue 4 January, 1982

In This Issue...

Renew Now, the Price is Going Up

If you renew your subscription before March 1, 1982, you can renew at the current rate of $12/year. Starting March 1st, the price will go up to $15/year (2nd class mail in the USA). Subscriptions sent First Class Mail to USA, Canada, and Mexico will be $18/year. Air Mail subscriptions to all other countries will be $28/year. The price for back issues will be $1.50 each (plus $1.00 postage outside of USA, Canada, and Mexico).

S-C MACRO Assembler II is almost here!

I am committed to having a finished product by February 15th. This is what I have been calling Version 5.0, but I have decided to call it S-C MACRO Assembler II instead. Version 4.0 will still be sold at $65. The MACRO version will be $80. Owners of Version 4.0 can upgrade for only $27.50. There will be an all new manual, rather than the current 2-part manual.

The MACRO Assembler includes macros (of course!), conditional assembly, EDIT, COPY, global string replacement, and many more new features. And it assembles even faster than version 4.0!

Hi-Res SCRN Function with Color.....David Doudna

I am a 15-year-old living in St. Louis, Missouri. While looking through the back issues of Apple Assembly Line, I found "Hi-Res SCRN Function for Applesoft" (May, 1981 issue). I noticed the routine only returned a 0 or 1, and you challenged readers to write one to return a color value 0-7. Well, I did it. My version is not interfaced to Applesoft; that is an exercise for the reader! (I use the Programmer's Aid ROM with FORTH.)

I am not going to explain how hi-res colors work, beyond the facts that two adjacent dots are white; the upper bit in each byte of the hi-res screen adds 4 to the color value; an isolated bit is color 1 or 2 (or 5 or 6) depending on the X-position. If you want to understand my program, you should study more about hi-res plotting first.

A word about the color value.... In Applesoft you specify color value with a number from 0 to 7. The Programmer's Aid ROM uses color values of 0, 42, 85, 127, 128, 170, 213, and 255. My program returns both numbers for the color: the 0-7 index in HCOLOR, and the P.A.ROM color value in COLOR.BYTE.

Lines 1060-1140 define the variables used; these are in the same locations as those used by the Programmer's Aid ROM. If you want to modify the program to work with Applesoft, be sure to put these variables in the correct locations. Two more variables are defined at lines 2120,2130.

Lines 1160-1180 pick up the X- and Y-coordinates. I assume you have stored the coordinates here before calling HSCRN. Lines 1190-1390 calculate the base address for the particular horizontal line your point is on. This code is just copied out of P.A.ROM. Lines 1410-1530 divide the X-coordinate by 7 to get the byte offset on the line. The quotient is left in the Y-register. The remainder is used to pick up a bit mask to select the particular bit within the byte.

Lines 1540-1650 make the first color check. The high-order bit of the byte (half-dot shift control). If the bit specified = zero, the color is black. If it = one, the color depends on whether either neighbor of this dot = one. If neither neighbor = one, the color depends on whether this dot is in an even or odd column. If the color is not black, I put 1 or 2 in the X-register to indicate the color it will be if it is not white.

Lines 1660-1790 check the neighbor bit on the left to see if it = one. Notice that there are several special cases. First, the left-neighbor might be in the same byte. Second, it might be in the byte to the left of this one. Third, there might not be a byte to the left of this one.

Lines 1800-1920 check the neighbor bit on the right. The same kind of special cases exist here, and they are handled the same way.

Line 1940 sets X = 3 for white color. Line 1960 gets the color

value in the A-register. All paths merge at line 1980, with the color index 0-3 in the A-register. All that remains is to add 4 if the half-dot shift control = 1 (Lines 1980-2010.

Lines 2020-2060 convert the color index to a color byte (by simple table-lookup), and return. Line 2100 is the table of color values.

Here is a table of colors (their names, index numbers, and P.A.ROM numbers):

		Color Byte Value		
Color	Index	Hex	Dec	Binary
BLACK	0	00	0	00000000
GREEN	1	2A	42	00101010
VIOLET	2	55	85	01010101
WHITE	3	7F	127	01111111
BLAC▼2	4	80	128	10000000
ORANGE	5	AA	170	10101010
BLUE	6	D5	213	11010101
WHITE2	7	FF	255	11111111

The program works with either page 1 or page 2 of Hi-Res. Set HPAGE to $20 for page 1, or $40 for page 2.

To call this program from Integer BASIC, you would first POKE the X- and Y-coordinates, then CALL the program, and then PEEK the color value. From assembly language, set up the coordinates and JSR HSCRN. The color index is returned in the X-register, and the color byte value in the A-register.

[Program and article modified somewhat by the editor]

```
                        1000 *--------------------------------
                        1010 *      HI-RES SCRN FUNCTION WITH COLOR
                        1020 *
                        1030 *      BY DAVID DOUDNA, FERGUSON, MISSOURI
                        1040 *      NOVEMBER 30, 1981
                        1050 *--------------------------------
0026-                   1060 HBASL  .EQ $26       BASE ADDRESS
0027-                   1070 HBASH  .EQ $27
0030-                   1080 HMASK  .EQ $30       BIT MASK
                        1090 *--------------------------------
0320-                   1100 XOL    .EQ $320      X-COORDINATE
0321-                   1110 XOH    .EQ $321
0322-                   1120 YO     .EQ $322      Y-COORDINATE
0324-                   1130 HCOLOR.BYTE .EQ $324
0326-                   1140 HPAGE  .EQ $326      HI-RES PAGE ($20 OR $40)
                        1150 *--------------------------------
0800- AD 22 03          1160 HSCRN  LDA YO        GET (A)=Y-COORDINATE
0803- AE 20 03          1170        LDX XOL       GET (Y,X)=X-COORD.
0806- AC 21 03          1180        LDY XOH
0809- 48                1190        PHA           Y-COORD BITS LABELED ABCDEFGH
080A- 29 C0             1200        AND #$C0      CALCULATE BASE ADDRESS FOR Y-COORD
080C- 85 26             1210        STA HBASL          IN HBASL,HBASH FOR
080E- 4A                1220        LSR                ACCESSING SCREEN MEMORY
080F- 4A                1230        LSR                VIA (HBASL),Y
0810- 05 26             1240        ORA HBASL     HBASH = PPPFGHCD
0812- 85 26             1250        STA HBASL     HBASL = EABAB000
0814- 68                1260        PLA           WHERE PPP=001 FOR $2000-3FFF
0815- 85 27             1270        STA HBASH       AND PPP=010 FOR $4000-5FFF
0817- 0A                1280        ASL
0818- 0A                1290        ASL
0819- 0A                1300        ASL
```

```
081A- 26 27     1310        ROL HBASH
081C- 0A        1320        ASL
081D- 26 27     1330        ROL HBASH
081F- 0A        1340        ASL
0820- 66 26     1350        ROR HBASL
0822- A5 27     1360        LDA HBASH
0824- 29 1F     1370        AND #$1F
0826- 0D 26 03  1380        ORA HPAGE
0829- 85 27     1390        STA HBASH
                1400 *-------------------------------
082B- 8A        1410        TXA             DIVIDE X-COORD BY 7 (7 DOTS PER BYTE)
082C- C0 00     1420        CPY #0          IS X-COORD > 255?
082E- F0 05     1430        BEQ .2          NO, ENTER SUBTRACTION LOOP
0830- A0 23     1440        LDY #35         YES: 256 = 7*36 + 4
0832- 69 04     1450        ADC #4          CARRY WAS SET, SO ADDS 5
                1460 *                      ALSO CLEARS CARRY, SO SBC #7 BELOW
                1470 *                      ACTUALLY SUBTRACTS 8
0834- C8        1480 .1     INY             INCREASE QUOTIENT
0835- E9 07     1490 .2     SBC #7          SUBTRACT 7 (OR 8 IF CARRY CLEAR)
0837- B0 FB     1500        BCS .1          STILL MORE 7'S
0839- AA        1510        TAX             REMAINDER IS BIT POSITION
083A- BD 9E 07  1520        LDA MSKTBL-249,X
083D- 85 30     1530        STA HMASK
                1540 *-------------------------------
083F- B1 26     1550        LDA (HBASL),Y     GET BYTE WHICH HAS OUR SPOT
0841- 29 80     1560        AND #$80        ISOLATE HALF-DOT SHIFT BIT
0843- 8D A6 08  1570        STA HIBIT
0846- B1 26     1580        LDA (HBASL),Y     GET BYTE AGAIN
0848- 25 30     1590        AND HMASK       ISOLATE OUR SPOT
084A- F0 38     1600        BEQ .9          COLOR IS BLACK (0 OR 4)
084C- AD 20 03  1610        LDA X0L         NOT BLACK
084F- A2 01     1620        LDX #1
0851- 4A        1630        LSR             ODD OR EVEN X-COORD.?
0852- B0 01     1640        BCS .3          ODD, COLOR=1 OR 5
0854- E8        1650        INX             EVEN, COLOR=2 OR 6
                1660 *-------------------------------
0855- A5 30     1670 .3     LDA HMASK       LOOK AT NEIGHBOR BIT ON LEFT
0857- 4A        1680        LSR             BITS ARE IN BYTE BACKWARDS
0858- 90 0D     1690        BCC .4          NEIGHBOR IN SAME BYTE
085A- 98        1700        TYA             NEIGHBOR IN DIFFERENT BYTE
085B- F0 0E     1710        BEQ .5          NO BYTE LEFT OF THIS ONE
085D- 88        1720        DEY
085E- B1 26     1730        LDA (HBASL),Y
```

```
0860- 29 40     1740          AND #$40
0862- D0 1D     1750          BNE .7          WHITE
0864- C8        1760          INY             RESTORE Y
0865- D0 04     1770          BNE .5          ...ALWAYS
0867- 31 26     1780 .4       AND (HBASL),Y
0869- D0 16     1790          BNE .7          WHITE
                1800 *--------------------------------
086B- A5 30     1810 .5       LDA HMASK       LOOK AT NEIGHBOR BIT ON RIGHT
086D- 0A        1820          ASL
086E- 10 0D     1830          BPL .6          NEIGHBOR IS IN SAME BYTE
0870- C0 27     1840          CPY #39         ALREADY AT RIGHT END?
0872- B0 0F     1850          BCS .8          YES, NOT WHITE THEN
0874- C8        1860          INY
0875- B1 26     1870          LDA (HBASL),Y
0877- 29 01     1880          AND #1
0879- D0 06     1890          BNE .7          WHITE
087B- F0 06     1900          BEQ .8          ...ALWAYS (NOT WHITE)
087D- 31 26     1910 .6       AND (HBASL),Y
087F- F0 02     1920          BEQ .8          NOT WHITE
                1930 *--------------------------------
0881- A2 03     1940 .7       LDX #3          COLOR IS WHITE (3 OR 7)
                1950 *--------------------------------
0883- 8A        1960 .8       TXA             COLOR TO A-REG
                1970 *--------------------------------
0884- 2C A6 08  1980 .9       BIT HIBIT       SEE IF HALF DOT SHIFT
0887- 10 03     1990          BPL .10         NO
0889- 18        2000          CLC
088A- 69 04     2010          ADC #4          YES
088C- 8D A7 08  2020 .10      STA HCOLOR
088F- AA        2030          TAX             USE COLOR # (0-7) TO GET COLOR BYTE
0890- BD 9E 08  2040          LDA COLOR.TABLE,X
0893- 8D 24 03  2050          STA HCOLOR.BYTE
0896- 60        2060          RTS
                2070 *--------------------------------
0897- 01 02 04
089A- 08 10 20
089D- 40        2080 MSKTBL .HS 01020408102040
                2090 *--------------------------------
089E- 00 2A 55
08A1- 7F 80 AA
08A4- D5 FF     2100 COLOR.TABLE .HS 002A557F80AAD5FF
                2110 *--------------------------------
08A6-           2120 HIBIT  .BS 1             MSB
08A7-           2130 HCOLOR .BS 1             COLOR INDEX 0-7
```

A Correction to "Step-Trace Utility"....Bob Sander-Cederlof

"Step-Trace Utility", published in the July 1981 issue of AAL (pages 17-20), has a bug. Three or four of you ran into the problem and called me about it, but I was never able to duplicate the problem. Finally Bob Leedom managed to pinpoint the bug, and I found out how to fix it.

If you have used Step-Trace, you might have noticed that it sometimes will hang-up or go crazy after a relative branch instruction. The problem is that if the 6502 was in decimal mode, the calculations are all incorrect. This affects the branch target, and also messes up screen output. To fix it, insert the following line:

```
2095        CLD          SELECT BINARY MODE
```

But how did the 6502 get into decimal mode, when I wasn't ever setting it? The contents of SAVE.P were random on initial start-up. Sometimes the contents managed to switch on decimal mode! Perhaps you should also insert the following two lines, to be certain of the initial status of the program you are tracing:

```
1455        LDA #0       CLEAR INITIAL STATUS
1456        STA SAVE.P
```

Future copies of Quarterly Disk #4 already have these two patches installed.

S-C ASSEMBLER II Version 4.0.....$55.00
Includes Manual, Diskette with Assembler and sample source programs, and Quick Reference Card.

Source code of Version 4.0 on disk.....$95.00
Fully commented, easy to understand and modify to your own tastes.

Cross Assembler Patches for 6809.....$20.00
Requires possession of Version 4.0. Enables you to develop programs for the Motorola 6809 CPU. (The MILL from Stellation, EXCEL-9 from ESD Laboratories, or the Radio Shack Color Computer.)

Cross Assembler for 6800.....$22.50
Requires possession of Version 4.0. Enables you to develop programs for the Motorola 6800, 6801, and 6802 CPUs.

AAL Quarterly Disks.....each $15.00
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.
QD#1: Oct - Dec 1980 QD#4: Jul - Sep 1981
QD#2: Jan - Mar 1981 QD#5: Oct - Dec 1981
QD#3: Apr - Jun 1981

Double Precision Floating Point for Applesoft.....$50.00
Provides 21-digit precision for Applesoft programs. Includes subroutines for standard math functions.

Some Simple Integer BASIC Games.....$10.00
Includes 4x4x4 tic-tac-toe, lo-res space war, lo-res jig-saw puzzle, and mastermind.

Blank Diskettes.....package of 20 for $50.00
Verbatim Datalife, with hub rings, no labels, in plain white jackets, in cellophane wrapper.

Lower-Case Display Encoder ROM.....$25.00
Works only Revision level 7 Apples. Replaces the encoder ROM. Comes with instructions.

Diskette Mailing Protectors.....10-99: 40 cents each
100 or more: 25 cents each
Corrugated folder specially designed for mailing mini-floppy diskettes. Fits in standard 6x9-inch envelope. (Envelopes 5-cents each, if you need them.)

Zip-Lock Bags (2-mil, 6"x9").....100 for $8.50
(2-mil, 9"x12").....100 for $13.00

Books, Books, Books.....compare our discount prices!

"Beneath Apple DOS", Worth & Lechner.....	($19.95)	$18.00
"What's Where in the Apple", William Leubert.....	($14.95)	$14.00
"6502 Assembly Language Programming", Leventhal..	($16.99)	$16.00
"Apple Assembly Language", Don & Kurt Inman.....	($12.95)	$12.00

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 We take Master Charge and VISA ***

6502 Relocator.....Bob Sander-Cederlof

Programs that are already assembled usually must be loaded at a specific memory address to execute properly. If you want to run it somewhere else, you have a problem. All the data references, JMP's, and JSR's will have to be examined to see if they need to be modified for the new location. If you don't have the source code, you can't re-assemble it. The other way, patching, can be quite a tedious operation!

Fortunately, way back in 1977, the WOZ (Steve Wozniak to you newcomers) wrote a program to do the work automatically. If you have the Programmer's Aid ROM then you have his RELOCATE program. You who have Apple II Plusses with the Language Card (also called 16K RAM card) can also use his program, because it is in the INTBASIC file along with Integer BASIC. (The latter group of people probably don't have the manual, though, because they didn't buy the ROM.)

I would like to see the RELOCATE program made more widely available, but it cannot be used as is unless you have Integer BASIC. Why? Because it uses SWEET-16 opcodes. RELOCATE also is itself tied to running at whatever location it is assembled for, so it can be a little trouble to find a place for it sometimes. By now you have probably guessed that I have recoded RELOCATE to solve both of these problems!

Paul Schlyter's article elsewhere in this issue of AAL shows RELOCATE put to good use. You can examine his instructions and learn most of what you need to know to use RELOCATE on your own programs. Basically, there are four steps:

1. Initialize. This sets up the control-Y monitor command. If RELOCATE is on a file, you do this with "BRUN RELOCATE".

2. Specify the program start and end addresses (where it now is in memory), and the new starting address (where you want it to be relocated to). This is done with the monitor command:

```
target<start.end^Y*
```

where "target" is the new starting address, and "start" and "end" are the addresses of the program where it is now. "^Y" means "control-Y". The "*" after the control-Y signals RELOCATE that you are in step 2 rather than step 3 or 4.

3. Specify the FIRST block to be copied "as-is" or to be "relocated" to the destination area. This is done with the monitor command:

```
    target<start.end^Y
or  target<start.endM
```

where "target" is the starting address in the new area for this block, and "start" and "end" define the block itself. Note that there is no trailing asterisk this time. Use

control-Y if you want this block relocated, or M if you want it copied as-is.

4. Specify the NEXT block to be copied as-is or relocated. You do this with the monitor command:

```
     .end^Y
or   .endM
```

where the target and start addresses are assumed to immediately follow the previously handled block, and "end" specifies the end of this new block. Use control-Y to relocate the block, or M to copy it as-is.

Obviously, step 4 above is repeated until the whole program has been copied/relocated. For each block of your program that is to be copied as-is, with no modification at all, you use the "M" command; for each block to be relocated you use the "control-Y" command.

If you need more detailed instructions and explanation, I must refer you to the manual. The Programmer's Aid #1 Manual is sold at most computer stores separately from the ROM package. Pages 11-28 explain why and how to use RELOCATE, and pages 80 and 81 contain the assembly listing.

Now here is my new version, which can be BRUN anywhere you have 134 ($86) bytes available. I have eliminated the SWEET-16 usage; this made the program slightly bigger, and a lot faster.

Lines 1260-1380 are the initialization code. They build the control-Y vector at $3F8-3FA. A JMP opcode is stored at $3F8; if you have DOS up this is redundant, but it won't hurt. Next I have to try to find myself. That is, where in memory am I (the program RELOCATE) located? JSR MON.RETURN (which is only an RTS instruction, so it comes right back without doing anything) puts the address of the third byte of the JSR instruction on the stack. Lines 1290-1370 use that address to compute the address of RELOC, and store it in $3F9 and $3FA.

When you type in a control-Y command, the monitor will now branch to RELOC at line 1400. Lines 1400-1430 look at the character after the control-Y in the command input buffer; if it is an asterisk, then you are trying to do step 2 above. If not, then you are on step 3 or 4. Lines 1440-1500 handle step 2, and lines 1510-1990 handle steps 3 and 4.

The part which used to be coded in SWEET-16 was lines 1690-1880. The SWEET-16 version took only 14 bytes, while the 6502 code takes 34 bytes. The 6502 version may take about 100 microseconds to execute, and the SWEET-16 version on the order of 1000 microseconds (for each instruction relocated).

```
                        1000 *-------------------------------
                        1010 *       6502 RELOCATION SUBROUTINE
                        1020 *-------------------------------
                        1030 *       MAY BE LOADED ANYWHERE, AS IT IS SELF-RELOCATABLE
                        1040 *-------------------------------
                        1050 *       ADAPTED FROM SIMILAR PROGRAM IN PROGRAMMERS AID #1
                        1060 *       ORIGINAL PROGRAM BY WOZ, 11-10-77
                        1070 *       ADAPTED BY BOB SANDER-CEDERLOF, 12-30-81
                        1080 *       (ELIMINATED USAGE OF SWEET-16)
                        1090 *-------------------------------
0034-                   1100 MON.YSAV   .EQ $34  COMMAND BUFFER POINTER
002F-                   1110 MON.LENGTH .EQ $2F  # BYTES IN INSTRUCTION - 1
F88E-                   1120 MON.INSDS2 .EQ $F88E  DISASSEMBLE (FIND LENGTH OF OPCODE)
FCB4-                   1130 MON.NXTA4  .EQ $FCB4     UPDATE POINTERS, TEST FOR END
FF58-                   1140 MON.RETURN .EQ $FF58
0100-                   1150 STACK      .EQ $0100     SYSTEM STACK
0200-                   1160 INBUF      .EQ $0200     COMMAND INPUT BUFFER
                        1170 *-------------------------------
003C-                   1180 A1     .EQ $3C,3D
003E-                   1190 A2     .EQ $3E,3F
0042-                   1200 A4     .EQ $42,43
0002-                   1210 R1     .EQ $02,03
0004-                   1220 R2     .EQ $04,05
0008-                   1230 R4     .EQ $08,09
000A-                   1240 INST   .EQ $0A,0B,0C
                        1250 *-------------------------------
0800- A9 4C             1260 START  LDA #$4C      JMP OPCODE
0802- 8D F8 03          1270        STA $3F8      BUILD CONTROL-Y VECTOR
0805- 20 58 FF          1280        JSR MON.RETURN    FIND OUT WHERE I AM FIRST
0808- BA                1290 START1 TSX
0809- CA                1300        DEX           POINT AT LOW BYTE
080A- 38                1310        SEC           +1
080B- BD 00 01          1320        LDA STACK,X  LOW BYTE OF START1-1
080E- 69 14             1330        ADC #RELOC-START1
0810- 8D F9 03          1340        STA $3F9
0813- BD 01 01          1350        LDA STACK+1,X   HIGH BYTE OF START1-1
0816- 69 00             1360        ADC /RELOC-START1
0818- 8D FA 03          1370        STA $3FA
081B- 60                1380        RTS
                        1390 *-------------------------------
081C- A4 34             1400 RELOC  LDY MON.YSAV COMMAND BUFFER POINTER
081E- B9 00 02          1410        LDA INBUF,Y  GET CHAR AFTER CONTROL-Y
0821- C9 AA             1420        CMP #$AA     IS IT "*"?
0823- D0 0C             1430        BNE RELOC2   NO, RELOCATE A BLOCK
0825- E6 34             1440        INC MON.YSAV YES, GET BLOCK DEFINITION
0827- A2 07             1450        LDX #7       COPY A1, A2, AND A4
0829- B5 3C             1460 .1     LDA A1,X
082B- 95 02             1470        STA R1,X
082D- CA                1480        DEX
082E- 10 F9             1490        BPL .1
0830- 60                1500        RTS
                        1510 *-------------------------------
0831- A0 02             1520 RELOC2 LDY #2       COPY NEXT 3 BYTES FOR MY USE
0833- B1 3C             1530 .1     LDA (A1),Y
0835- 99 0A 00          1540        STA INST,Y
0838- 88                1550        DEY
0839- 10 F8             1560        BPL .1
083B- 20 8E F8          1570        JSR MON.INSDS2  GET LENGTH OF INSTRUCTION
083E- A6 2F             1580        LDX MON.LENGTH  0=1 BYTE, 1=2 BYTES, 2=3 BYTES
0840- F0 31             1590        BEQ .3       1-BYTE OPCODE
0842- CA                1600        DEX
0843- D0 0C             1610        BNE .2       3-BYTE OPCODE
0845- A5 0A             1620        LDA INST     2-BYTE OPCODE
0847- 29 0D             1630        AND #$0D     SEE IF ZERO-PAGE MODE
0849- F0 28             1640        BEQ .3       NO (X0 OR X2 OPCODE)
084B- 29 08             1650        AND #$08
084D- D0 24             1660        BNE .3       NO (80-FF OPCODE)
084F- 85 0C             1670        STA INST+2   CLEAR HIGH BYTE OF ADDRESS FIELD
                        1680 *-------------------------------
0851- A5 04             1690 .2     LDA R2       COMPARE ADDR TO END OF SOURCE BLOCK
0853- C5 0B             1700        CMP INST+1
0855- A5 05             1710        LDA R2+1
0857- E5 ??             1720        SBC INST+2
0859- 90 ?8             1730        BCC .3       ADDR > SRCEND
085B- 38                1740        SEC          COMPARE ADDR TO BEGINNING OF SRC
085C- A5 0B             1750        LDA INST+1
085E- E5 02             1760        SBC R1
0860- A8                1770        TAY
0861- A5 0C             1780        LDA INST+2
0863- E5 03             1790        SBC R1+1
0865- 90 0C             1800        BCC .3       ADDR < SRCBEG
```

```
0867- AA       1810         TAX
0868- 98       1820         TYA           ADDR = ADDR-SRCBEG+DESTBEG
0869- 18       1830         CLC
086A- 65 08    1840         ADC R4
086C- 85 0B    1850         STA INST+1
086E- 8A       1860         TXA
086F- 65 09    1870         ADC R4+1
0871- 85 0C    1880         STA INST+2
               1890 *-------------------------------
0873- A2 00    1900 .3      LDX #0        COPY MODIFIED INSTRUCTION TO DESTINATION
0875- A0 00    1910         LDY #0
0877- B5 0A    1920 .4      LDA INST,X    NEXT BYTE OF THIS INSTRUCTION
0879- 91 42    1930         STA (A4),Y
087B- E8       1940         INX
087C- 20 B4 FC 1950         JSR MON.NXTA4    ADVANCE A1 AND A4, TEST FOR END
087F- C6 2F    1960         DEC MON.LENGTH  TEST FOR END OF THIS INSTRUCTION
0881- 10 F4    1970         BPL .4        MORE IN THIS INSTRUCTION
0883- 90 AC    1980         BCC RELOC2    END OF SOURCE BLOCK
0885- 60       1990         RTS
```

A Review of THE INDEX.....Bob Sander-Cederlof

THE INDEX is a new book that you can use. No doubt you subscribe to three or more magazines and newsletters, out of the 100 or so that are being published with information Apple owners want and need. Wouldn't you like a composite index that covered the best ones?

Bill Wallace an attorney in St. Louis, Missouri, has put together just such an index. His book compiles over 12000 articles, editorials, and columns from over 900 issues of personal computer magazines published during the last six years. Over 40 different magazines and newsletters are covered. I am honored that Bill has chosen to include both of my newsletters: Apple Assembly Line, and AppleGram.

Organized as a Key-Word in Context (KWIC) index, there are over 30000 entries. There are 92 pages of Apple-related articles, 160 pages covering other computers (Apple owners will be interested in the CP/M and 6502 sections), and over 200 pages of general articles. All the information necessary for obtaining copies and/or subscriptions of the various magazines and newsletters is also included.

Bill plans to publish a second edition later this year to include the issues published since the cutoff date of the first edition, as well as lots of additional publications that were not previously covered.

THE INDEX costs $14.95, and is available from Missouri Indexing, Inc., P. O. Box 301, St. Ann, MO 63074. Bill is responsive to requests for group rates, if you can interest your local Apple club; call him at (314) 997-6470.

Serious Problem in Apple DOS.....Bob Sander-Cederlof

If you are trying to use the IRQ interrupt line for any purpose, and also DOS. you may have run across this problem before. Apparently at random, for no good reason, you may get the NO BUFFERS AVAILABLE message.

The reason is that both DOS and the IRQ interrupt code are trying to use the same page zero location: $0045. DOS uses this location as part of a pointer address when looking for the next available buffer. (See the code at $A2CB-A2CF and $A764-A780.) DOS also uses $0045 when printing the catalog (see $ADB9, $AE09, and $AE53).

The IRQ interrupt code in the Apple Monitor ROM (at $FA86) uses $0045 to save the contents of the A-register. If an interrupt occurs while DOS is in the process of looking for a buffer, POW!

One solution is to turn off interrupts whenever DOS may be active, using the SEI opcode. A better solution would be quite difficult: look through all of DOS and modify every reference to $0045 (or to $0044 and $0045 as a pair) to use some other location in page zero. A third possible solution for those who can do it is to modify the Apple Monitor ROM to use some other location to save the A-register.

In case you ARE using interrupts and DOS together, you should also know that RWTS does inhibit interrupts while it is active. After a call to RWTS is complete, the interrupt-inhibit status is restored to whatever it was before the call. Interrupts cannot be allowed during RWTS, because of the critical software timing code involved in reading and writing the disk.

----------(C E E M A C)----------

THE VISUAL COMPOSITION SYSTEM

--

RELEASE 1.0 NOW AVAILABLE

$40*

CEEMAC IS THE LANGUAGE OF FIRE ORGAN, WHICH WAS THE 1ST 'SOFT VISUAL ALBUM'.

A SYSTEM TO CREATE PERFORMANCE LEVEL ABSTRACT VISUALS TO ACCOMPANY MUSIC HAS ELUDED OUR GRASP FOR EONS. MY PERSONAL SEARCH HAS LED TO CEEMAC.

DESIGNED PRIMARILY TO SUPPORT THIS NEW ART FORM, CEEMAC PROVIDES A HIGHLY INTERACTIVE COMPOSING ENVIRONMENT. IT REQUIRES AN APPLE][COMPUTER RUNNING DOS 3.3 WITH 48K OF RAM MEMORY.

THE LANGUAGE COMES WITH A FIFTY PAGE MANUAL AND OVER FORTY SCORES INCLUDING THE ORIGINAL 36 FROM FIRE ORGAN. THEY HAVE BEEN SPECIALLY COMMENTED BY THE COMPOSERS TO HELP YOU GET STARTED.

--

* ADD NOTHING FOR TAXES, POSTAGE, HANDLING, ETC, ANYWHERE IN THE USA

VAGABONDO ENTERPRISES
1300 E ALGONQUIN - 3G
SCHAUMBURG IL - 60195

(DIRECT MAIL ORDERS ONLY)

Putting S-C Assembler II on the Language Card . . Paul Schlyter

[Paul is a subscriber in Stockholm. Sweden.]

Introduction

I have owned the S-C Assembler II for only a little more than three weeks, and already I have stopped using the two other assemblers I used to use before ("Apple Text Processing System" and "DOS Tool Kit Assembler"). Although the others have some powerful features, the S-C Assembler is so much easier to use it now takes me only about half the time to finish an assembly language program as it did before.

The many similarities between the S-C Assembler and Integer BASIC made me curious, so I disassembled the Assembler. Earlier I have done the same thing with Integer BASIC, Applesoft, and DOS. It wasn't too long (about a week) that I had a fair understanding of the first third of the assembler. Then the idea turned up in my head: "Why not try to relocate it into the language card?" Another week of sleepless nights and it was up and running!

There were several traps on the way. It took a long time for me to discover the address stack put into DOS at $1333-133C. Sometimes I just entered the regular assembler at $1000-24FF and didn't notice anything, sometimes the machine crashed when a DOS error occurred. But that was a week ago, and the last week nothing like that has happened...now I feel fairly confident that I have found all bytes that need to be relocated. If anything does turn up, I will let you know.

Why and How

Have you ever thought about how very similar to Integer BASIC the S-C Assembler II is? It stores its source files as DOS type-I files, and numbers the lines the same way as Integer BASIC. Just like in Integer BASIC, you have access to all DOS commands. Well, the similarities don't stop there. Integer BASIC starts at address $E000 and ends a little bit above $F400; the S-C Assembler starts at $1000 and ends a little bit above $2400. The byte at $E000 is $20 in Integer BASIC (JSR opcode) while it is $4C in Applesoft (JMP opcode); it is by looking at this byte that DOS decides whether Integer BASIC or Applesoft is the current language. Well, guess what the byte at $1000 in the S-C Assembler is: it's $20!

When putting all these facts together, I started to wonder if it wasn't possible to relocate the S-C Assembler up into the Language card, making DOS believe it is in Integer BASIC. This of course requires that you have Applesoft on the motherboard ROMs, so that DOS will be able to distinguish between the ROM and Language card languages.

Sure enough, it is possible. I did move it up there, and it works, and it turned out to be really convenient. The DOS

command FP puts me in Applesoft, while INT puts me into the S-C Assembler! Also, if I am currently in Applesoft and LOAD an S-C Assembler source file (type-I, of course), DOS will automatically start up the assembler! Can you really ask for more?

To relocate the S-C Assembler into the language card, you need of course a language card (any of the several RAM cards no available will do). You also need to have Applesoft in ROM on the motherboard (not Integer BASIC). You also need a relocation program; I used the one in the Programmer's Aid #1 software, which is in the INTBASIC file on the DOS 3.3 System Master. You could use the one in Bob Sander-Cederlof's article elsewhere in this AAL just as well.

Step-by-step Procedure

1. Boot the DOS 3.3 System Master. This will load Integer BASIC into the Language Card.

2. Type INT to enter Integer BASIC.

3. Put in the S-C Assembler II disk, and BLOAD ASMDISK 4.0 (do not BRUN it).

4. Enter the Apple monitor by typing CALL -151. (Throughout the following steps, be sure you do NOT hit RESET!)

5. Now that you are in the monitor, type the following commands:

```
C083 C083           (write-enable the language card)

D4D5G               (initialize the relocation program)

E000<1000.24FF^Y*   (specify source and destination
                     blocks for the relocation program.
                     Note that "^Y" means "control-Y".
                     The asterisk at the end IS necessary.)

E000<1000.100E^Y    (relocate the first segment)

.100FM .121E^Y .1282M .1330^Y .133CM .1436^Y

.1438M .147C^Y .14A9M .14DB^Y .141EM .14F3^Y

.14F5M .15D0^Y .15D6M .17A6^Y .17AEM .1A8C^Y

.1A91M .1BB7^Y .1CAEM .2149^Y .2150M .221C^Y .24FFM
```

(The monitor commands on the above four lines relocate the program segments and move the data segments. They can be typed as shown, or one per line, or even all on one line. Just be sure to type them correctly -- check and double check -- before pressing RETURN.)

6. The machine code relocator automatically updates any direct address references in the program being relocated. This saves us a lot of work, but it does not finish the work. We also have to fix all the address tables and all immediate address references. Enter the following monitor commands to fix all of these (only one command per line):

```
E042:E2     E254:E2     E334:E0     F234:F1
E227:E7     E259:EB     E336:E0     F239:F0
E22C:E5     E25E:E3     E338:E0     F23E:F0
E231:E0     E263:E0     E33A:E3     F243:F0
E236:E5     E268:E5     E33C:E0     F248:F2
E23B:E1     E26D:F1     E4A3:E4     F24D:F0
E240:E4     E272:E6     E83E:F2     F252:F1
E245:E1     E277:E1     F225:F0     F257:F0
E24A:E6     E27C:E0     F22A:F1     F25C:EE
E24F:E3     E281:EB     F22F:F0     F261:E0
```

7. The cold start routine in the Assembler must be patched:

```
E030:ED E2

E2E0:AD 83 C0 AD 83 C0 A9 00 85 D9 4C 08 E3 AD 83 C0

E2F0:AD 83 C0 4C 75 E3
```

8. If you wish, you may change the starting address of the Assembler Symbol Table to make more space:

```
E011:10

E2D6:10
```

9. If you enter Applesoft from the S-C Assembler, the output hook from DOS will still be connected to the S-C Assembler output routine. But the assembler will be banked away since now the motherboard ROMs are enabled! The result is that the Apple will hang. To cure this problem, you will have to sacrifice the SLOW and FAST commands, and the ability to suspend/abort listings using the space bar and RETURN keys. This is not such a big sacrifice anyway, since all language card owners have the Autostart Monitor: you can use control-S to suspend a listing. You can also use RESET to abort one (provided your language card has a switch and it is in the UP position). [If you can't bear to part with SLOW and FAST, you can type FP and then hit RESET to get out of the Assembler.]

Here are the patches to eliminate SLOW and FAST:

```
E1E9:EA EA EA EA

E22D:4D 4E 54 68 FF

E273:FF FF FF
```

These patches also change FAST to MNT, a command that gracefully enters the monitor. From the monitor, a control-C will re-enter the S-C Assembler with the current source program intact; a control-B will cold start the S-C Assembler.

10. Save the completed package on disk with:

```
BSAVE LANGASM,A$E000,L$2000.
```

11. Modify a copy of the HELLO program from the DOS 3.3 System Master Disk to BLOAD LANGASM instead of INTBASIC, and use this as your HELLO program. When you boot it will automatically load the S-C Assembler II into your language card.

Parting Shots

Maybe you think that I must have a thorough knowledge of how the S-C Assembler II works internally to be able to do this relocation, but this is not actually the case. I made a disassembly and also hex and ASCII dumps of the whole assembler, and I also started to untangle the code, but I only really know about a third of the code fairly well. I still have not the faintest idea of how the actual assembly is performed, although looking at the ASCII dump immediately revealed where the opcode and command tables were located, and the error messages. I also did find out the places where the error messages are produced...this helps a lot in figuring out what is happening in the code. And with this not-too-well understanding of the inner workings, and with a lot of trial-and-error, I was able to find all the places where changes needed to be made.

My S-C Assembler has been running from my language card for over a week, and I have used it a lot during this time; all has gone very well. And believe me, it is SO CONVENIENT to have it there! I really benefit from the language card, not only when using Pascal or CP/M, but also when I am running DOS. And I use the S-C Assembler II much more than Integer BASIC, so having the assembler in the language card is really the right thing for me. Maybe it is for you too!

So, Bob, although you have made an excellent and very easy to use assembler, it is not quite true anymore that the S-C Assembler II is the easiest assembler to use...LANGASM is. And as you have guessed, LANGASM is nothing but the S-C Assembler II relocated into the language card!

[If the instructions for making LANGASM leave you breathless, you can order Quarterly Disk #6 ($15.00). It will include all the source code from this issue and the next two of AAL, and also an EXEC file which will create LANGASM from ASMDISK 4.0. It will be ready in early March, 1982. Another shortcut is to order the source code of the S-C Assembler II ($95.00). Then simply change the origin, modify the SLOW and FAST commands, and re-assemble it. Voila! LANGASM!]

Handy EXEC Files.....Bob Sander-Cederlof

Now that I have my Firmware card with Integer BASIC on it plugged into slot 4, I am all too frequently needing to fix those two bytes in DOS. For some reason I don't get around to putting the patched DOS onto every disk. But with a few EXEC files I can make the patches very easily now.

The first EXEC file, which I call INT, is like this:

```
CALL -151           (get into the monitor)
C081                (turn off the language card, if on)
C0C1                (turn off the firmware card, if on)
A5B8:C0             (patch DOS to use firmware card)
A5C0:C1
3D3G                (return to DOS and Applesoft)
INT                 (enter Integer BASIC)
```

The second file I use to load LANGASM into the Language Card (see Paul Schlyter's article elsewhere in this issue of AAL). Here is what it looks like:

```
CALL-151            (get into the monitor)
C0C1                (turn off the firmware card, if on)
C081 C081           (write enable the language card)
BLOAD LANGASM       (load LANGASM into the language card)
A5B8:80             (patch DOS to use the language card)
A5C0:81
3D3G                (return to DOS and Applesoft)
INT                 (enter the assembler)
```

The third EXEC file I use to patch DOS back to its normal mode of using the language card in slot 0. If I have already loaded the S-C Assembler II (LANGASM) into that card, but was using Integer BASIC, EXEC ASM will get me back to the assembler.

```
CALL-151            (get into the monitor)
C081                (turn off the language card, if on)
C0C1                (turn off the firmware card, if on)
A5B8:80             (patch DOS to use the language card)
A5C0:81
3D3G                (return to DOS and Applesoft)
INT                 (enter the assembler)
```

Just for fun, here is one more EXEC file. This one copies the contents of the firmware card in slot 4 into the language card in slot 0. A much faster way of loading it with Integer BASIC than running HELLO on the DOS 3.3 System Master!

```
CALL-151            (get into the monitor)
C0C0                (turn on the firmware card)
1000<D000.FFFFM     (copy firmware card into mother RAM)
C0C1                (turn off the firmware card)
C081 C081           (write enable language card)
D000<1000.3FFFM     (copy stuff into the language card)
3D0G                (return to DOS)
```

If you don't have an editor that will help you build EXEC files like these, here is a short Applesoft program which will do it. I have also included a short program to display the file, in case you need to do that.

Both of these programs CALL 64874. which is the Apple Monitor subroutine to read a line into the system buffer starting at $200. The CALL -3288 in READ EXEC FILE is to fix the ONERR stack pointer.

```
100  REM WRITE EXEC FILE
130 D$ =  CHR$ (4): INPUT "FILE NAME: ";F$
140  PRINT D$"OPEN"F$: PRINT D$"DELETE"F$
150  PRINT D$"OPEN"F$: PRINT D$"WRITE"F$
160  GOSUB 500: IF  PEEK (512) = 175 AND  PEEK (513) = 170 AND  PEEK
     (514) = 141 THEN 220
170 I = 511
180 I = I + 1:C =  PEEK (I): PRINT  CHR$ (C);: IF C <  > 141 THEN
     180
190  GOTO 160
220  PRINT D$"CLOSE"
230  END
500  REM INPUT A LINE WITHOUT DOS KNOWING
505  IN# 0: PR# 0: CALL 64874: CALL 1002: RETURN
```

```
100  REM READ EXEC FILE
130 D$ =  CHR$ (4): INPUT "FILE NAME: ";F$
140  PRINT D$"NOMONCIO": PRINT D$"OPEN"F$: PRINT D$"READ"F$
150  ONERR  GOTO 220
160  CALL 64874
170 I = 511
180 I = I + 1:C =  PEEK (I): PRINT  CHR$ (C);: IF C <  > 141 THEN
     180
190  GOTO 160
220  CALL  - 3288: PRINT D$"CLOSE"
```

6500/1 One-Chip Computer.....Dan Pote

Commodore Semiconductor Group has announced a new one-chip microcomputer, called the 6500/1. (I believe the same chip is available from Synertek and Rockwell.) The 6500/1 has a 6502 CPU and is compatible with existing 6502 programs. There are also four I/O ports (32 bi-directional lines, the equivalent of two 6522 devices), a counter, 2048 bytes of ROM, and 64 bytes of static RAM. Your choice of 1- or 2-MHz internal clock. It can be ordered as masked-ROM, PROM, or piggy-back EPROM. For more information call Commodore at (214) 387-0006.

A New Compiler.....Bobby Deen

For about 3 months now I have been using test versions of a new compiler from Laumer Research. The compiler is called FLASH! and compiles Integer Basic programs into fast machine language programs. The machine code will execute 12 times faster than Integer Basic and the speed increase gets even greater on larger programs.

This compiler unlike all others on the market today has the ability to produce not only binary machine code but will make assembly language listings and files. Complete with labels for line numbers, forward references, variables, and runtime package entry points. All assembly language is compatable with the S-C Assembler II syntax and the file format is the familiar 'I' type DOS file ready to assemble with the runtime package!

This does save time in the development of machine language programs! A program which might take 2 weeks of your time to develop in assembly language, can be done in 2 hours in basic then compiled to get the speed benifit of machine language. If you have the runtime source code option you can take an assembly language file that the compiler can produce and, using the S-C Assembler, you can edit the inner loops of you program to gain even more speed or add new assembly language modules of your own!

The runtime package I keep referring to is the collection of subroutines which support the compiled code by providing commonly used functions such as routines to multiply and divide numbers, string handling functions, and input/output routines. The runtime package is self contained and does not require Integer Basic to allow a compiled program to execute (hark ye software authors!). The compiled code will even execute on a cassette based system if you ask FLASH! to put the runtime code into the code file with your compiled program. FLASH! will not normally include the runtime package in your output file to conserve your valuable disk space. Instead it will use a standard runtime package that runs from $800-$1BFF and is loaded by a short loader placed at the beginning of the output file.

FLASH! will compile every normal Integer Basic statement except the statement "LIST" which is ignored by FLASH!. FLASH! also has 28 new statements and 3 new functions which provide DATA, READ, HPLOT, DRAW, XDRAW, CHR$, 16 bit PEEK's and POKE's, hex input and output abilities, hex strings of almost any length (to free memory size), and more to provide almost everything that could be of use in a systems programming language.

I have a hires graphics program which generates mazes that I have wanted to code in machine language for some time now. In Basic the program takes about 12 minutes to run. I go to high school during the daytime and I am in the school band. After school I have two part time jobs to take up most of the rest of my time. So its been hard to find time to convert the program to machine language with so little spare time.

Once I got a test version of FLASH! and compiled the maze program it would draw a maze in 1.25 minutes! A dramatic increase in program speed. Using the assembly languge file output by FLASH! I recoded some of the code more efficiently and got the maze program to run in 55 seconds! Now it almost takes longer to print the maze on my printer than it does to run the compiled version of the maze program.

The FLASH! compiler is so easy to use that I have been using it for three months before the preliminary reference manual was printed. FLASH! will keep track of a compilation by status fields on the bottom line of the Apple II screen. Since a compilation can take several minutes the status information is handy to have. I think that the FLASH! compiler is one of the slickest programs in my program library and really compliments the S-C Assembler II. I plan much use for it in the future.

Here is an example of the code produced by the FLASH! compiler.

for the basic statements :

```
10 A=3
20 B=-1000*A
30 PRINT B/(A+2)
```

the following code is created by FLASH! :

```
1C00-                    .OR $1C00        program origin
                         .IN R.RUNTIME LOADER
1C1E- 20 63 0A L.START   JSR R.INIT       initialize runtime package
1C21- 00 00              .DA LINE.TBL     line number table address
1C23- 00 00              .DA LINE .TBLN   number of entries
1C25- 00 00              .DA DATA.TBL     data statement table
1C27- 00 00              .DA DATA.TBLN    number of entries
1C29- 78 1C              .DA DSP.VAR.TBL  begining of DSP variables
1C2B- 7D 1C              .DA END.VARS     end of all variables
1C2D- 7D 1C              .DA FREE.P       free memory pointer
1C2F- 83 79              .DA FREE.S       size of free memory
1C31- 00                 .HS 00           flag for error routine
1C32- A9 03    L.10      LDA #3           lo byte of constant 3
1C34- A0 00              LDY /3           hi byte of constant 3
1C36- 8D 79 1C           STA A            store lo byte of A
1C39- 8C 7A 1C           STY A+1          store hi byte of A
1C3C- A9 18    L.20      LDA #-1000       load lo byte of -1000
1C3E- A0 FC              LDY /-1000       load hi byte of -1000
1C40- 20 39 09           JSR R.PSH        push -1000 on stack
1C43- AD 79 1C           LDA A            get lo byte of A
1C46- AC 7A 1C           LDY A+1          get hi byte of A
1C49- 20 9B 09           JSR R.MUL1       multiply -1000*A
1C4C- A9 7B              LDA #B           lo adrs of B
1C4E- A0 1C              LDY /B           hi adrs of B
1C50- 20 2B 13           JSR R.ASG.VAR    save value into B
1C53- AD 7B 1C L.20      LDA B            load lo byte of B
1C56- AC 7C 1C           LDY B+1          load hi byte of B
1C59- 20 39 09           JSR R.PSH        push value of B on stack
```

```
1C5C- AD 79 1C          LDA A              load lo byte of A
1C5F- AC 7A 1C          LDY A+1            load hi byte of A
1C62- 20 39 09          JSR R.PSH          push value of A on stack
1C65- A9 02             LDA #2             get lo byte of 2
1C67- A0 00             LDY /2             get hi byte of 2
1C69- 20 48 09          JSR R.ADD1         compute A+2
1C6C- 20 D0 09          JSR R.DIV2         compute B/(A+2)
1C6F- 20 2B 12          JSR R.PRTN1        print result
1C72- 20 8E FD          JSR MON.CROUT      print car retn
1C75- 4C B9 0D          JMP R.NOEND        go to NOEND error
                LINE.TBL .EQ 0             no table because no GOTO's
                LINE.TBLN .EQ 0            with expressions on them
                DATA.TBL .EQ 0             no table because no
                DATA TBLN .EQ 0            data statements
1C78- 00        DSP.VAR.TBL .HS 00         end of DSP variables
1C79-           A       .BS 2              2 byte value of A
1C7B-           B       .BS 2              2 byte value of B
1C7D-           END.VARS                   end of variable area
1C7D-           FREE.P                     start of free memory
                FREE.S  .EQ 31107          size of free memory
1C7D-                   .EN                end of code
```


Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $12 per year in the U.S.A., Canada, and Mexico. Other countries add $12/year for extra postage. Back issues are available for $1.20 each (other countries add $1 per back issue for postage).